VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Konzistence operací v distribuovaných databázových systémech

Operation Consistency in Distributed Database Systems

2013 Petr Pejcel

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Petr Pejcel**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Konzistence operací v distribuovaných databázových systémech
Operation Consistency in Distributed Database Systems

Zásady pro vypracování:

Některé distribuované databáze jako Voldemort nebo DynamoDB umožňují u jednotlivých operací nastavit vyžadovaný typ konzistence a tím ovlivnit propustnost databáze. Cílem práce je prozkoumat možnosti dnešních distribuovaných databází z pohledu nastavení úrovně konzistence jednotlivých operací a jejich porovnání.

Práce bude mít následující části:
1. Rešerše databázových systémů podporující různou konzistenci operací.
2. Vytvoření sady testů pro otestování úrovně konzistence databáze.
3. Výběr dvou databázových systémů a jejich porovnání s využitím vytvořených testů.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího diplomové práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Radim Bača, Ph.D.**

Datum zadání: 16.11.2012

Datum odevzdání: 07.05.2013

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2013 .....

Rád bych na tomto místě poděkoval Ing. Radimovi Bačovi, Ph.D., protože bez něj by tato práce nevznikla.

Abstrakt

Tato diplomová práce se zabývá problematikou konzistence operací v distribuovaných NO-SQL databázových systémech. NO-SQL databáze jsou nový koncept vyvinutý pro nahrazení relačních databází v určitých případech užití. V úvodních kapitolách je popsáno co je to NO-SQL databázový systém a jaké přináší výhody. Další kapitoly popisují principy a techniky, které NO-SQL databáze obecně používají. Dále pak popisuje možnosti jak ovlivnit konzistenci operací u NO-SQL databází a záverečné kapitoly se zabývají testováním propustnosti NO-SQL databází Cassandra a Riak.

Klíčová slova: NO-SQL, MapReduce, ACID, CAP teorém, BASE, key-value store, document store, graph store, eventuální konzistence, vektorové hodiny, DynamoDB, Cassandra, Riak.

Abstract

This thesis deals with the operation consistency in NO-SQL distributed systems. NO-SQL database is a new concept develeped to replace relational databases in particular use cases. The first chapters describe what is NO-SQL database system what benefits it brings. Next chapters describe the principles and techniques that NO-SQL databases use in general. Then it describes the possibilities to influence the consistency of the operations in NO-SQL databases and final chapters are concerned with testing throughput of NO-SQL databases Cassandra and Riak.

Keywords: NO-SQL, MapReduce, ACID, CAP theorem, BASE, key-value store, document store, graph store, eventual consistency, vector clock, DynamoDB, Cassandra, Riak.

Seznam použitých zkratek a symbolů

SQL	–	Structured Query Language
NO-SQL	–	Not Only SQL
SŘBD	–	Systém Řízení Báze Dat
ACID	–	Atomicity,Consistency, Isolation, Durability
BASE	–	Basically available, Soft-state, Eventual consistency
CAP	–	Consistency, Availability, Partition tolerance
LDAP	–	Lightweight Directory Access Protocol
DNS	–	Domain Name System
RAC	–	Real Application Cluster
SAN	–	Storage Area Network
JSON	–	JavaScript Object Notation
XML	–	Extensible Markup Language
RDF	–	Resource Description Framework
PHP	–	Hypertext Preprocessor
EC2	–	Elastic Compute Cloud
AWS	–	Amazon Web Services
CQL	–	Cassandra Query Language
MB	–	Mega Bajt
VPN	–	Virtual Private Network

Obsah

Seznam tabulek

Seznam obrázků

1 Úvod

V dnešní době je téměř celý svět propojen internetem. Když vezmeme v potaz sociální sítě je použití internetu ještě mnohem rozsáhlejší. S postupným růstem využití internetu se také zvětšuje objem uložených dat reprezentujících obsah. Tento růst tak vede k nebývalé potřebě škálovatelnosti platforem, které zpracovávají a uchovávají tato data. Tradičním postupem v situaci, kdy je nutné zvýšit výkon systému, je pořízovaní stále výkonějšího hardwaru dokud není dosaženo potřebného výkonu. Tento postup se však stále častěji ukazuje jako nedostačující a proto byly vyvinuty tzv. NO-SQL distribuované databázové systémy.

1.1 Stručný obsah jednotlivých kapitol

Kapitola 2 popisuje obecně co je to NO-SQL databáze, její základní charakteristiky a rozdíly mezi NO-SQL databází a tradičními relačními databázemi. Kapitola 3 se zabývá základními principy a technikami použitými v NO-SQL databázích. Kapitola 4 obsahuje rešerši databázových systému, které podporují nastavení různé úrovně konzistence operací. Kapitoly 5 a 6 se zabývají testováním propustnosti databází Cassandra a Riak. Propustnost je zkoumána na základě nastavení úrovně konzistence pro jednotlivé operace. Kapitola 7 popisuje testování řízeného výpadku uzlu při vytížení databáze. Kapitola 8 obsahuje porovnání databází Cassandra a Riak z pohledu propustnosti. V závěrečné kapitole jsou shrnuty výsledky a cíle, které byly dosaženy při řešení této diplomové práce a návrh dalšího vývoje této diplomové práce.

2 Co je a proč nasadit NO-SQL databázi

Nejen významné společnosti se zabývají problematikou škálovatelnosti při použití tradičních relačních databází. Použití relačních databází však může být neschůdným řešením jak po finanční tak provozní stránce. Například Google byl nucen použít čtyřicet tisíc MySQL instancí a Facebook dokonce musel utrácet jeden milión amerických dolarů měsíčně za použití speciálního databázového hardwaru [1]. Tento hardware sloužil především ke zpracování fotek uživatelů. Tyto neživotaschopná řešení tak vedly k přehodnocení stávajících databázových technologií a nasměrovaly vývoj k NO-SQL řešením.

2.1 Co je NO-SQL databáze

Samotná zkratka NO-SQL skrývá pojem NotOnly SQL, tedy ne pouze SQL. Pojem NO-SQL byl poprvé zaveden v roce 1998 a označil tak relační databázi, kde nebyl použit SQL jazyk [5]. V roce 2009 pak byla vyslovena myšlenka, že vývoj NO-SQL bude smyslem pro získávání alternativ a že bude muset řešit problémy, které relační databáze vyřešit schopny nejsou [5]. Mnoho startup projektů především pro aplikace Webu 2.0 pak bylo odstartováno bez nasazení databáze Oracle či MySQL, které byly do té doby hojně využívány. Oracle a MySQL byly nahrazeny novými databázovými systémy, které konceptuálně vycházely z DynamoDB a Google Bigtable. Tyto databázové servery a jejich projekty se pak staly projekty pod open source licencí. Například databáze Cassandra (4.2) byla původně vyvíjena společností Facebook a nyní je open source řešením v rámci Apace Software Project. Nutno podotknout, že velice úspěšným řešením - Cassandra dokáže zapisovat 2500 krát rychleji do databáze o velikosti 50GB než MySQL [5].

2.2 Základní charakteristiky NO-SQL databází

Základní charakteristiky, které mají NO-SQL databáze jsou:

- Jsou spuštěny a běží na velkém množství uzlů,
- data jsou rozděleny a replikovány mezi uzly,
- NO-SQL databáze nedodržují striktní konzistenci dat (více v pak v kapitole 2.4).

Z těchto charakteristik pak vyplývají dva hlavní důvody k jejich nasazení [1]:

- Automatická replikace dat mezi více uzly, jinými slovy distribuovaný databázový systém,
- všechny uzly se podílejí na zpracovávání požadavků - škálování výkonu.

Většina NO-SQL databází používá pro ukládání dat vysoce výkonný souborový systém a programovací model MapReduce vyvinutý společností Google pro paralelní zpracování objemných dat. MapReduce je zjevně složen ze slov map a reduce, jinými slovy mapuj a redukuj. Každé z těchto slov označuje hlavní úkoly a principy MapReduce.

Obecně je úkolem mapování zpracovat množinu dat a převést ji na novou množinu, kde jsou data reprezentována dvojicemi klíč - hodnota. Vstupem redukce je tedy výstup mapování a výstupem je nová, menší množina dvojic. Pořadí jednotlivých kroků není zaměnitelné. Důvodem takového zpracování většinou bývá snaha optimalizovat danou množinu dat, popřípadě z ní získat relevenatní informace.

Pro zpracování velkého množství dat využívá MapReduce velké množství uzlů souhrně označených jako klastr (cluster), tedy množina uzlů. Označení klastr se používá jestliže jsou uzly umístěny v místní síti. Pro uzly rozmístěné v zeměpisně odlišných lokalitách je pak seskupení uzlů označováno jako grid. V každé z těchto množin je pak uzel (může jich však být více) v roli master a uzly v roli slave (3.8, ovšem u některých NO-SQL řešení tento fakt neplatí, protože nevyužívají architekturu master-slave.

2.3 Rozdíly mezi NO-SQL a tradičními relačními databázemi

NO-SQL řešení se oproti tradičním SQL databázím liší dle konkrétních potřeb pro uchovávání a zpracování dat. Především pak nepoužívá relační datový model a ACID. Více o ACID v kapitole 2.4.1. Základní rozdíly jsou následující[1]:

- V každé aplikaci není nutné explicitně ukládat vazby mezi daty,
- přenesení vazeb mezi daty mimo databázový systém umožňuje rozšíření databáze podle potřeby,
- díky absenci předdefinovaného databázového schématu je možné jednoduše rozšířit datový model,
- jelikož nejsou ukládany vazby mezi daty, zpracování operací čtení a zápisu je pak mnohem jednodušší, právě díky absenci vazeb mezi daty.

Přenesení zoodpovědnosti za korelaci souvisejích dat, řešení konfliktů a integritních omezení na programovou vrstvu umožňuje NO-SQL systémům dosáhnout požadovaného výkonu a škálovatelnosti. Toto přenesení však vyžaduje obrovské zkušenosti a úsilí při návrhu NO-SQL řešení. Usílí pro přechod na NO-SQL databázi tak především spočívá v tom, že ne každý programátor je schopen přejít ze světa relačních databází, které má již zažité. Realita ukazuje, že přechod na NO-SQL databázi je užitečný především pro aplikace, jejichž hlavním úkolem je zpracování velkého objemu dat.

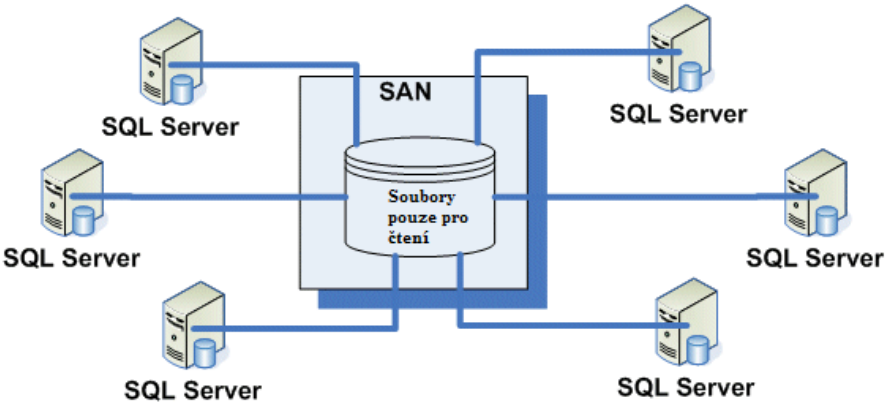
2.3.1 Škálovatelnost SQL databází

Škálovatelností se obecně rozumí schopnost aplikace využít efektivně více zdrojů ke zvýšení výkonu aplikace. Pro představu aplikace běžící na jedno procesorovém serveru je schopná obsloužit čtyři uživatele. Pokud stejná aplikace dokáže na čtyř procesorovém systému obsloužit patnáct uživatelů, pak se dá považovat za škálovatelnou [2]. Taková škálovatelnost se označuje jako vertikální. Škálovatelnost je obecně u relačních databází

omezena především množinou vlastností ACID, integritními omezeními a vazbami mezi daty.

2.3.1.1 SQL server nabízí několik alternativ k řešení škálovatelnosti. Pro porovnání s NO-SQL řešeními budou popsány následující dvě [2]:

1. Sdílené databáze
2. Peer to peer replikace

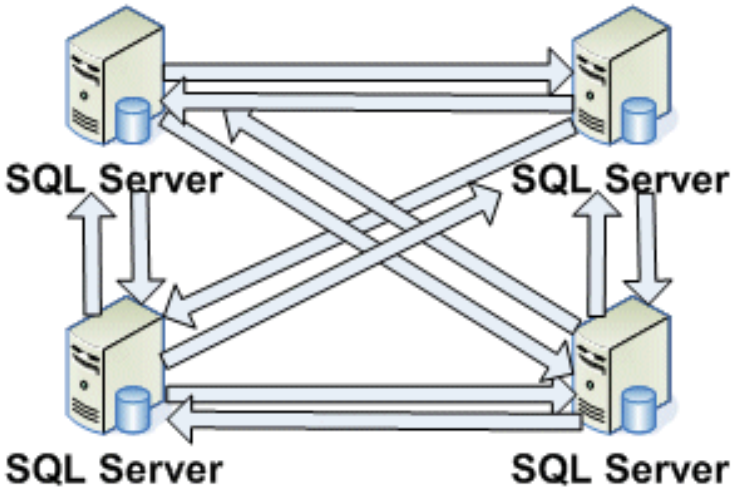


Obrázek 1: Schéma sdílené databáze [2]

Sdílené databáze - je nejjednodušší řešení škálovatelnosti SQL serveru. Schéma takového řešení obsahuje jedno centrální úložiště - ve většině případů databázi a k ní připojených až osm databázových serverů, které poskytují data. Jedná se totiž pouze o read-only řešení, to znamená, že centrální úložiště není za běhu aplikace aktualizováno. Připojené instance tedy slouží pouze k zobrazování dat uložených v centrálním úložišti a ta navíc musí být spuštěna v prostředí SAN sítě. Schéma takové sítě je zobrazeno na obrázku 1.

Jasnou výhodou tohoto řešení je rozdělení zatížení mezi jednotlivé servery. Nevýhodou je jistě to, že se jedná pouze o read-only řešení a je tedy vhodné např. pro datové sklady či reportovací účely [2]. Pokud je potřeba data v centrálním úložišti aktualizovat, tak je nutné ostatní databáze odpojit a zvolit jednu, která data aktualizuje. Je nutné však brát v potaz, že tato aktualizace může trvat dlouhou dobu pokud centrální úložiště uchovává velké množství dat a je nutné také velké množství údajů aktualizovat, proto se v konkrétních případech spouští centrální úložiště dvě. Zatímco jedno je aktualizováno, druhé slouží dále svému účelu. Limit osmi databází není technický limit, ale limit testovaný společností Microsoft. U dalších verzí SQL serveru bude tento limit vyšší [2].

Peer to peer replikace - je relativně novou technologií SQL serveru a oproti sdílené databázi nabízí řešení škálovatelnosti u dat, která jsou v průběhu času aktualizována, avšak aktualizace nejsou tak časté jako u běžných aplikací. V tomto řešení každý server spravuje vlastní kopii. Replikace je zde využita k propagaci změn v datech. V tomto případě jsou změny v jedné databázi šířeny do všech ostatních databází. Problém však nastává při řešení konfliktů, které toto řešení nepodporuje. Tento problém byl vyřešen zavedením množiny pravidel zvané data stewardship, tedy jakési správcovství dat. Tyto pravidla říkají, že aktualizace databáze, tedy zápis, může provádět pouze server, který databázi spravuje, čili vlastník. Tato replikace je tedy použitelná pokud konflikty nenastávají. Další nevýhodou je, že replikace dat databáze každého uzlu je replikována do všech ostatních uzlů. Což může být problém pokud by bylo potřeba použít peer to peer replikaci mezi větším množstvím uzlů. Schéma peer to peer replikace je znázorněno na obrázku 2.



Obrázek 2: Schéma peer to peer replikace [2]

2.3.1.2 Oracle RAC Řešení škálovatelnosti pomocí Oralce RAC se přibližuje řešení škálovatelnosti u NO-SQL databází, více o škálovatelnosti NO-SQL databází se nachází v kapitole 2. Uzly v jednom klastru se pro aplikace či klienty jeví jako jedna jediná databáze. Oracle RAC může obsahovat až sto instancí Oracle databáze, které sdílejí jednu databázi. Mezi jednotlivými instancemi je pak rovnoměrně rozdělena zátěž databáze. RAC nepodporuje spuštění klastru ve více datových centrech. Nabízí pouze extended distance cluster, což je záloha původní clusteru při využití zrcadlení diskových polí. Navíc tato záloha se musí nacházet v datovém centru maximálně v rámci města a tím pádem nemusí pokrýt všechny možné příčiny výpadků datacentra jako jsou například povodně či tornáda, při kterých by mohly být vyřazeny obě datacentra najednou.

Nasazení Oracle RAC však má mnoho podmínek a omezení. Jedna z těchto podmínek je, že RAC musí být spuštěn na Oracle Clusterware, což je software, který umožňuje právě vytvoření klastru složeného z nezávislých uzlů. RAC je tzv. Shared everything databáze. Všechny datové soubory, logy apod. musí být uložený na sdílených clusterware discích. Hlavní výhodou Oracle RAC je zvýšení dostupnosti aplikace, protože se nemůže stát, že jeden server se potýká s výpadkem a aplikace není funkční [4]. Nevýhodou je cena licence Oracle RAC, která činí 23 000 amerických dolarů [3].

2.3.2 Škálovatelnost NO-SQL databází

Jak bylo zmíněno výše, zpracování velkého objemu dat může být problém pro relační databázi.

Oproti SQL databázím nabízí NO-SQL databáze jednoduchá, časově nenáročná, levná řešení, která navíc nejsou omezena vlastnostmi ACID. Spuštění NO-SQL databáze je nenáročné na znalost příslušné databáze. Spuštění klastru nevyžaduje téměř žádné předchozí znalosti. Společnosti vyvíjející tyto databáze nabízí stručné a dobře zpracované návody ke spuštění databáze a to at'už se jedná o spuštění jedné instance, nebo celého klastru.

Škálovatelností se v tomto případě rozumí, že je databáze škálovatelná horizontálně. To znamená, že není nutné fyzicky přidat hardware, aby bylo možné spustit další instanci databáze. Navíc spuštění nové instance NO-SQL databáze je otázkou i několika minut (záleží na konkrétní databázi). Navíc spuštění nové instance (přidání nového uzlu) může být provedeno aniž by byla databáze odstavena (neplatí však vždy).

2.4 CAP teorém

CAP teorém v zásadě popisuje rozdíly přístupu k rozmístění aplikační logiky u NO-SQL databází oproti klasickým relačním databázím. Zkratka CAP vyjadřuje:

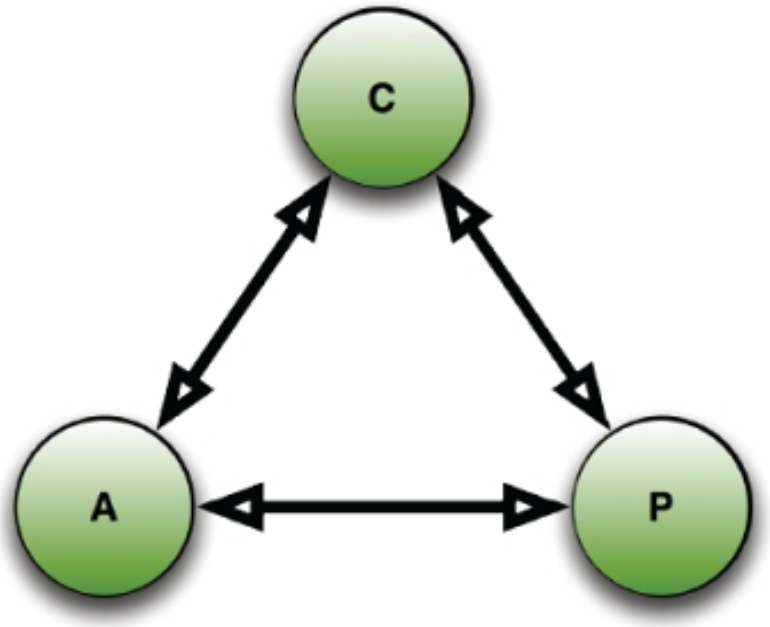
- Consistency
- Availability
- Partition tolerance

Consistency - konzistence popisuje jestli a kdy je systém konzistentní po provedení databázové operace. Obecně jsou NO-SQL databáze považovány za konzistentní, jestliže po provedení operace zápis, vidí výsledek tohoto zápisu všichni, kdo provedou operaci čtení.

Availability - dostupnost vyjadřuje, že systém je navržen a implementován tak, aby umožňoval funkčnost (dotazování) i když uzly v clusteru zaznamenají výpadek, či jsou vypnuty například při výměně hardwaru.

Partition tolerance - tolerance vůči výpadkům symbolizuje odolnost databázového systému v případech, když nastanou výpadky sítě, či internetového připojení. Tento výpadek tak znamená zamezení v komunikaci jednotlivých uzlů systému. Z jiného pohledu se dá také chápat jako schopnost systému pracovat při dynamickém přidávání nebo odebírání uzlů v klastru. Oproti SQL řešením, také může znamenat výpadek celé datacentra.

V reálném světě však nelze dosáhnout všech tří vlastností najednou. Pro určitý případ použití může být důležitější, aby byl systém konzistentní a odolný vůči výpadkům. V takovém případě je vhodnější použít databázi splňující pravidla ACID. Jestliže je důležitější dostupnost společně s tolerancí vůči výpadkům pak je vhodnější použít databázi charakterizovanou skupinou vlastností BASE - viz kapitola 2.4.1. Dalším případem pak je kombinace vlastností, kde je systém dostupný a odolný vůči výpadkům. Pro lepší viditelnosti jsou tyto kombinace zobrazeny na obrázku 3. Písmena C, A a P označují jednotlivé vlastnosti - Consistency, Availability a Partition tolerance.



Obrázek 3: Schéma peer to peer replikace [6]

Pojmy ACID a BASE jsou vysvětleny v kapitole 2.4.1. Tabulka 1 zobrazuje jednotlivé kombinace možností a příklady využití [5].

2.4.1 ACID vs BASE

Relační SŘBD jsou charakteristické využitím modelu ACID. ACID je množina vlastností popisující databázové transakce a jejich vlastnosti. Zkratka ACID znamená:

Kombinace	Příklady
konzistence + dostupnost	LDAP, jednoduchý web
konzistence + tolerance vůči výpadkům	distribuované databáze, distribuované zamykání
Dostupnost + odolnost vůči výpadkům	DNS, Coda, NO-SQL

Tabulka 1: Kombinace vlastností CAP teorému

- Atomicity
- Consistency
- Isolation
- Durability

Atomicity - atomicita popisuje databázovou transakci jako nedělitelnou operaci. Jinými slovy se buď provede celá, nebo se neprovede vůbec.

Consistency - konzistence zaručuje, že se databáze, po provedení jakékoliv transakce, přenese z jednoho korektního (konzistentního) stavu do druhého. Konzistence také zaručuje zapsání validních dat do databáze také vzhledem k vazbám a integritním omezením.

Isolation - izolovanost zaručuje, že jsou transakce bezpečně a nezávisle zpracovány současně a bez vzájemných konfliktů. Nezaručuje však pořadí operací.

Durability- trvalost znamená, že jakmile je databázová transakce dokončena, tak její výsledek je trvale uložen v databázi a nemůže být ztracen například kvůli výpadku elektrické energie, nebo pádu systému.

Soubor vlastností ACID zaručuje silnou konzistenci. V rozsáhlejších aplikacích je však konzistence dosaženo za cenu dostupnosti. V mnoha aplikacích , jako například na sociáních sítích, je však důležitější dostupnost než konzistence a to ve smyslu, že je vidět, že se něco děje, ale není to plně konzistentní. Proto je zaveden soubor vlastností BASE popisující vlastnosti NO-SQL databází. Zkratka BASE vyjadřuje:

- Basically Available
- Soft-state
- Eventual consistency

Basically available - v podstatě k dispozici znamená, že systém poskytuje dostupnost z hlediska CAP teorému.

Soft-state - naznačuje, že stav systému se může měnit v průběhu času a to i bez vstupních dat. Je to způsobeno např. replikací dat apod. V průběhu tohoto času se data přibližují ke konzistentnímu stavu.

Eventual consistency - eventuální konzistence vyjadřuje, že data postupně konvergují ke konzistetnímu stavu.

BASE model se dá shrnout následovně, aplikace beží v podstatě pořád (basically available), data nemusí být vždy konzistentní (soft state), ale je v eventuélně známém stavu (eventual consistency) [5]. Vlastnosti ACID a BASE jsou porovnány v tabulce 2.

ACID	**BASE**
Silná konzistence	Nízká konzistence
Izolovanost transakcí	Dostupnost na prvním místě
Soustředěnost na "commit"	Best effort
Vnořené transakce	Jednodušší
Dostupnost	Rychlejší
Obtížný vývoj (z pohledu schématu)	Jednoduchý vývoj (z pohledu schématu)

Tabulka 2: Srovnání vlastností ACID a BASE

Model BASE není vhodný pro každé řešení, ale je určitě vhodnou alternativou k řešení modelem ACID, které nepotřebuje dodržovat striktně relační model.

3 Základní principy NO-SQL databází

Každá z NO-SQL databází má svá vlastní specifika. Následující kapitola popisuje obecné principy a rozdělení technologií NO-SQL databází. Jednotlivé databáze, především ty, které podporují nastavení různé úrovně konzistence, budou popsány detailněji v kapitole 4.

3.1 Základní rozdělení NO-SQL databází

NO-SQL databáze se dají rozdělit podle mnoha vlastností, nejjednodušeji se dají rozdělit podle datového modelu:

- Key-value store,
- document store,
- graph store.

Key-value store - data jsou zde reprezentována jako množina dvojice klíč hodnota (množina hodnot) a hodnota může mít libovolný formát či délku, nezáleží zde, jestli hodnota bude obsahovat jedno neboo deset polí. Touto reprezentací připomínají key-value databáze tradiční relační databáze. S tím rozdílem, že není nutné, aby všechny záznamy v dané množině (neuvádím zde název - liší se u konkrétních databází, bude popsáno v kapitole 4) měly stejnou délku, tedy počet atributů. Také není nutné, aby atributy měly hodnotu - samotný název atributu se dá považovat jako hodnota. Tato reprezentace je podobná například datovému typu vocabulary (slovník) v programovacím jazyce Python. Key-value databáze nepodporují vazby mezi daty. Zástupci dokumentově orientovaných databází jsou DynamoDB, Riak a další.

Document store - neboli dokumentově orientované úložiště. Jak samotný název napovídá data v dokumentově orientovaných databázích jsou popsána jako dvojice klíč-hodnota (jako u key-value store), ale hodnota je zde reprezentována jako dokument. Hodnoty zde mohou být indexovány a mohou na nich být prováděny rozsahové dotazy [8]. Neexistují zde tedy žádné tabulky jako v relačních databázích. Databáze tedy neobsahuje žádné řádky, buňky či vazby mezi tabulkami. Taková databáze se dá nazvat také jako schema-less. Výhodou oproti relačním databázím je, že pokud je nutné doplnit u dokumentu (neboli záznamu) atribut, tak se jednoduše přidá. Není nutné uchovávat tento atribut prázdný či null jako v relační databázi.

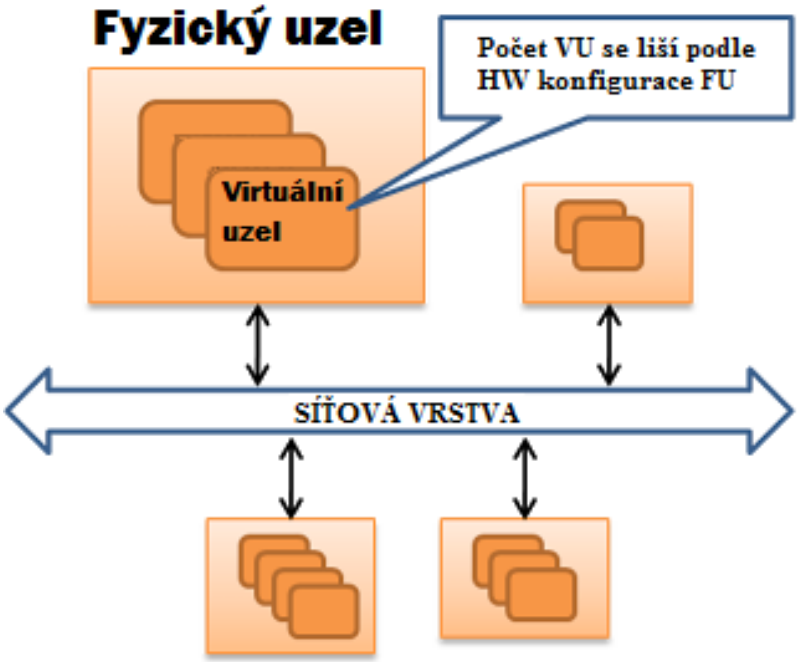
Jednotlivé dokumenty uložené v dokumentově orientované databázi jsou nezávislé, což zvyšuje výkon databáze a snižuje vedlejší účinky souběžnosti. Většina dokumentových databází také podporuje verzování dokumentů a to jako built-in funkci. Při dotazování dokumentové databáze jsou dokumenty reprezentovány jako JSON a v některých případech i jako XML objekty. Zástupci dokumentově orientovaných databází jsou CouchDB, RavenDB, MongoDB a další. [7]

Graph store - grafové databáze vychází z teorie grafů. Data jsou zde reprezentována jako trojice - uzel, hrana a vlasnost. Jsou používoný mimojiné pro reprezentaci RDF objektů v ontologiích sémantického webu nebo dat na sociálních sítích. Typickým zástupcem grafových databází je Neo4j.

3.2 Infrastruktura NO-SQL databáze

Základní infrastruktura je složena z mnoha (stovky až tisíce) nespolehlivých uzlů propojených sítí. Nespolehlivost je myšlena tím, že není zaručena jejich funkčnost v čase. Každý stroj se nazývá fyzický uzel. Každý z uzlů má stejnou softwarovou konfiguraci, ale může se lišit hardwarovou konfigurací - procesor, paměť, pevný disk. Na každém fyzickém uzlu pak běží určitý počet virtuálních uzlů, počet se odvíjí samozřejmě vzhledem k hardwarové konfiguraci fyzického uzlu a požadavkům příslušné databáze [9].

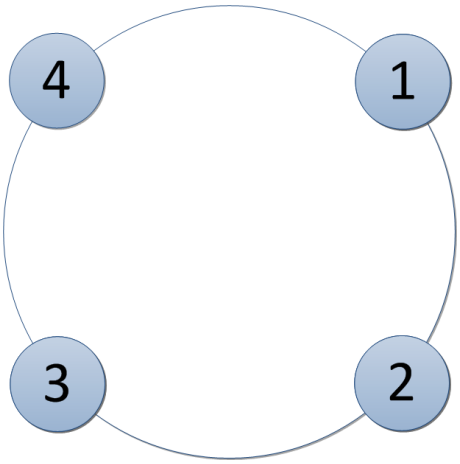
Hlavní myšlenkou rozložení virtuálních uzlů je, aby na stejném fyzickém uzlu nebyly spuštěny virtuální uzly, které jsou zoodpovědné za stejná data (3.4). Pokud tedy vypadne virtuální uzel není funkcionalita systému téměř ovlivněna a není třeba provádět zvláštní opatření. V případě výpadku fyzického uzlu je nutné rozdělit jeho zátěž mezi několik dalších fyzických uzlů. Schéma infrastruktury je znázorněno obrázku 4.



Obrázek 4: Schéma infrastruktury NO-SQL databáze [9]

3.3 Schéma NO-SQL databáze

Jednotlivé uzly v NO-SQL databázi jsou umístěny v klastru. Klastr se dá chápat jako kružnice na jejímž obvodu jednotlivé uzly leží. Pořadí a rozložení uzlů v klastru je u NO-SQL databází provedeno tak, aby byly uzly rozloženy rovnoměrně po obvodu kružnice. Rovnoměrné rozložení je potřeba proto, aby byla rovnoměrně rozložena zátěž mezi uzly. Schéma klastru při jeho spuštění je na obrázku 6.



Obrázek 5: Schéma klastru NO-SQL databáze

3.4 Rozdělení dat

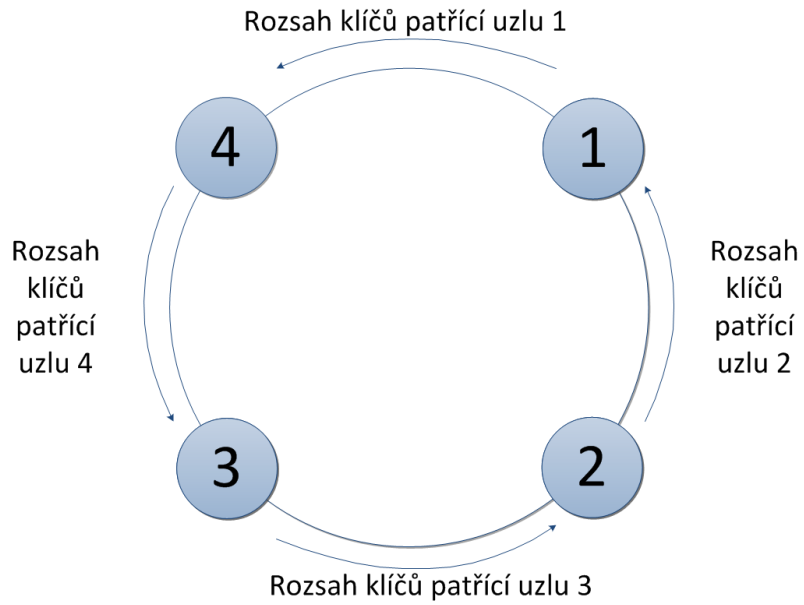
Rozdělení dat (data partitioning) se zabývá rovnoměrným rozdělením dat mezi zoodpovědné uzly. Nejjednodušším způsobem by bylo určit příslušnost podle rovnice:

$$prislusnost \quad = \quad hasovaciFunkce(klic) \ \ mod \ \ pocetUzlu$$

Výsledkem takového výpočtu by pak byl klastr jako na obrázku 6. Výstupem hašovací funkce může být číslo, či řetězec. Pro jednoduchost uvedu příklad s čísly. Jestliže je výsledek hašovací funkce celé číslo pak je příslušnost uzlu zřejmá. Pokud výsledek hašovací funkce leží např. v intervalu (1, 2> pak se nachází mezi uzly 1 a 2 (dle obrázku 6. Příslušnost daných dat se pak určí po směru hodinových ručiček, tedy za příslušný klíč bude zoodpovědný uzel 2. Analogicky se pak postupuje u dalších intervalů.

Problém u tohoto způsobu rozdělení dat nastává ve chvíli, kdy je často měněn počet uzlů, ať už z důvodu výpadků nebo přidávání uzlu(ů) do klastru. Příslušnost jednotlivých klíčů se pak dramaticky mění a je potřeba redistribuce dat.

Pro odstranění tohoto problému bylo zavedeno tzv. konzistentní hašování, které stanoví pevné rozložení klíčů po obvodu kružnice. U každé databáze je potřeba určit rozsah



Obrázek 6: Rozdělení dat v klastru

klíčů (key range), což je konečná množina, obsahující všechny možné klíče (v závislosti na datovém typu - řetězec, číslo...). Označení klíč tady výchází z key-value datového modelu, kde všechna uložená data (hodnota, dokument, graf) mají svůj klíč. Pro každý klíč je pak určena hodnota (příslušnost), která vyjadřuje zoodpovědnost uzlu za příslušná data. Výpočet tedy vypadá takto:

$$prislusnost \quad = \quad hasovaciFunkce(klic)$$

Je zřejmé, že je vyřešen problém s přidáním či odebráním uzlu a není nutná redistribuce dat v rámci celého klastru, ale pouze mezi uzly zoodpovědnými za příslušná data.

3.5 Změna struktury klastru

NO-SQL databáze poskytují další nedílnou výhodu a to, že je možné přidávat či odebírat uzly do/z klastru, aniž by byla ovlivněna funkcionalita dotčeného klastru.

3.5.1 Přidání uzlu do klastru

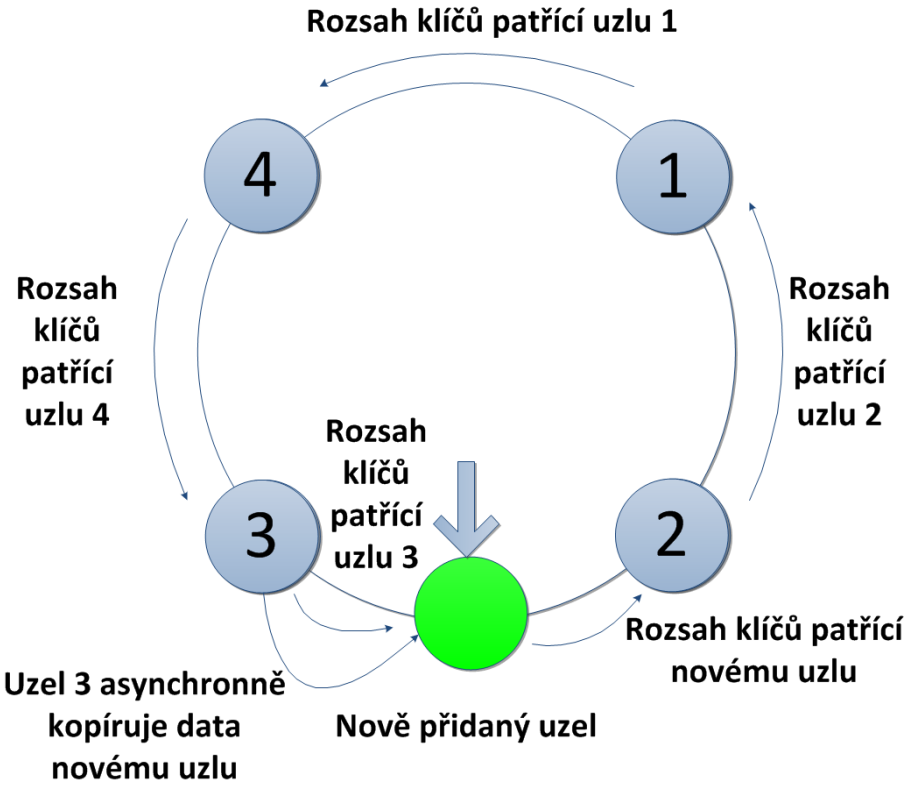
Přidání uzlu do klastru probíhá následovně:

1. Příslušnému uzlu je pomocí konzistentního hašování určena pozice v klastru.
2. Uzel oznámí svou přítomnost a jeho sousedům (po a proti směru hodinovýc ručiček) je synchroně upravena zoodpovědnost za příslušné klíče a replikovaná data.

3. Nový uzel začne pomocí hromadných operací kopírovat data, od příslušného souseda, za která je nyní zoodpovědný, toto již probíhá asynchronně.

4. Změna v klastru je asynchronně rozšířena ostatním uzlům.

Protože oznámení o změně v klastru je šířeno asynchronně, je možné, že některé uzly o této změně ještě neví a mohou replikovat či přeposílat požadavky data na uzel, který už není za tato data zoodpovědný. Tento problém je však vyřešen v druhém kroku přidávání nového uzlu a tyto uzly pak požadavky a replikovaná data přeposílají novému uzlu. Na druhou stranu nový uzel nemusí být stále připraven zpracovávat požadavky, protože nemá ještě dokopírována všechna data a tyto data mohou být v průběhu času zároveň aktualizována na původním uzlu. V této situaci se používají tzv. vektorové hodiny (kapitola 3.10) k určení zda je nový uzel již schopen zpracovávat požadavky. Pokud stále není požadavky jsou směrovány k původnímu uzlu.

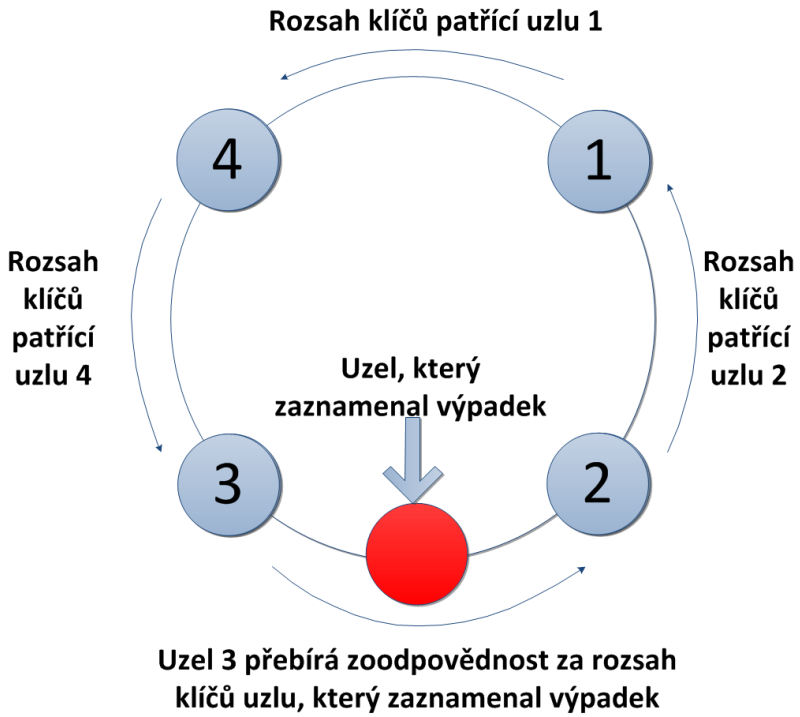


Obrázek 7: Přidání uzlu do klastru

3.5.2 Odebrání uzlu z klastru

Odebrání uzlu z klastru ať už z důvodu výpadku nebo úmyslného odpojení probíhá takto:

1. Odebíraný uzel neodpovídá ostatním uzlům na gossip (3.5.2.1) zprávy.
2. Sousedům je upravena asynchronně zoodpovědnost za příslušné klíče a replikovaná data.



Obrázek 8: Odebrání uzlu z klastru

3.5.2.1 Gossip protokol Gossip protokol v distribuovaných databázových systémech slouží k jednoduchému šíření periodických zpráv mezi jednotlivými uzly. V případě NO-SQL databází se používá především pro komunikaci mezi jednotlivými uzly v rámci klastru. Jednotlivé uzly tak ví, jestli jejich sousedé jsou funkční a také rozpoznávají nové uzly přidané do klastru [5].

3.6 Replikace dat

Replikace dat je nejdůležitějším principem NO-SQL databází ke zvýšení dostupnosti dat. Stupeň replikace určuje parametr *Replication factor* (replikační faktor. Tento parametr určuje, kolik uzlů bude udržovat kopii stejných dat. Replikace dat nejen zvyšuje dostupnost dat, ale také pomáhá ke snížení zátěže jednotlivých uzlů. Pokud je zatížen uzel 1 a uzly 2, 3 jsou téměř bez zátěže, požadavky na data udržována uzlem 1 mohou být přesměrovány na uzly 2 a 3. Tento příklad vychází z nastavení replikačního faktoru rovno 3.

Důležité je dodat, že v tomto případě se jedná o požadavky čtení. Požadavkům na aktualizaci musí být věnována větší pozornost, protože tyto požadavky je nutné směrovat na uzel udržující tyto data.

3.7 Konzistence na straně klienta

Jak bylo řečeno v kapitole 3.6, NO-SQL databáze uchovávají více kopií stejných dat. Je tedy nutné definovat jak je synchronizovat, aby klient, který se na databázi dotazuje, obržel konzistentní data. NO-SQL databáze nabízí několik modelů konzistence:

1. Strict consistency.
2. Read your write consistency.
3. Session consistency.
4. Monotonic read consistency.
5. Eventual consistency.

Strict consistency - striktní konzistence jako u relačních databází, v tomto modelu NO-SQL databáze uchovává pouze jednu kopii dat. Jinými slovy jestliže klient provede zápis či aktualizaci dat, tak ostatní klienti tuto změnu vidí okamžitě.

Read your write consistency - neboli čti svůj zápis znamená, že jakmile klient provede operaci zápisu, tak ji vidí okamžitě. Nevidí však výsledky operací jiných klientů i v případě, že je jeho dotaz na stejná data přesměrován na jinou repliku.

Session consistency - stejná jako v předchozím případě, avšak klient vidí pouze zápisy provedené jím samotným a to v rámci jedné relace.

Monotonic read consistency - jinými slovy monotóní čtění znamená, že klient uvidí vždy poslední verzi aktualizovaných dat.

Eventual consistency - eventuální konzistence znamená, že klient může vidět nekonzistentní verzi dat, čili starší verzi, protože změny jsou teprve replikovány mezi ostatní repliky. Tento model je užitečný pokud není častý souběžný přístup ke stejným datům a není tak nutné okamžitě vidět poslední verzi dat.

V závislosti na modelu, je nutné určit následující mechanismy:

- Jak budou požadavky přesměrovány k jednotlivým replikám,
- jak jednotlivé repliky propagují a aplikují změny v datech.

3.8 Master - slave model

Jak samotný název modelu napovídá vystupují zde repliky v rolích master a slave. Všechna data jsou v NO-SQL databázích rozdělena mezi více replik. Každou část těchto dat pak spravuje v tomto modelu replika v roli master a více replik v roli slave. Všechny operace update musí být nejprve směřovány k replice v roli master a jakmile je na ní update úspěšný, musí být tato změna rozšířena asynchronně k ostatním replikám. Existuje však možnost, kdy uzel master zaznamená výpadek, předtím než je změna rozšířena ke všem replikám. Proto se replika master snaží nejprve synchronně rozšířit k alespoň jedné replice slave.

Naproti tomu operace čtení mohou být směrovány k jakékoliv replice spravující data, avšak v závislosti na konzistenci, kterou klient požaduje. To, že požadavek může být směrován k jakékoliv replice je výhodou vzhledem k rozdělení zátěže mezi více replik. Pokud však klient požaduje striktní konzistenci, musí být požadavek směrován na repliku v roli master.

Rozdělení rolí replik v tomto modelu probíhá na úrovní virtuální. Neexistuje tedy fyzický uzel, který by byl v roli master. Každý fyzický uzel může tedy obsahovat několik replik v roli master i slave. Důležité je, že ne všechny repliky musí spravovat stejná data, dokonce nemusí být součastí stejného klastru. Tento fakt také přispívá k rozdělení zátěže mezi fyzické uzly. V případě výpadku fyzického uzlu může být ztracena replika v roli master, v tomto případě se stává masterem replika, jejíž data jsou nejčastěji aktualizována.

Master slave model je obecně vhodné použít pokud je aplikace zatížena vysokým počtem požadavků (čtení/zápis). Jedná se o nejčastěji používáný model v rámci NO-SQL databází [9].

Repliky v roli master používají dva modely jak šířit aktualizace k replikám slave:

- State transfer model,
- operation transfer model.

State transfer - zde master šíří k replikám slave stav (state) svých dat. Jakmile tento stav obdrží replika slave přepíše svůj aktuální stav na stav nový. V rámci propagace stavu není nutné posílat kompletní data, protože často nedochází ke změně objektu ve velké míře. Šířením kompletního stavu by zbytečně snižovalo propustnost sítě. Master tedy zjišťuje, která část dat byla změněna a pouze tato část je odeslána. Každý objekt je proto rozdělen na menší bloky a každému bloku je vytvořen verzovací strom. Poté master porovná jednotlivé verzovací stromy a zjistí, které bloky dat je potřeba odeslat [9].

Operation transfer - master šíří k replikám slave sekvenci operací, které byly na replice master provedeny. Sekvence operací se určuje podle chronologicky podle časové značky. Tyto operace jsou pak provedeny na každé replice. Při šíření operací je obecně přeneseno

méně dat, než při šíření stavu, což zvyšuje propustnost databáze. Avšak je nutné zaručit spolehlivé doručení konkrétní sekvence.

3.9 Multi master model

V případě, že je vysoká zátěž na určitý rozsah dat, master - slave model nedokáže rozdělit zátěž rovnovměrně. Všechny požadavky na aktualizace musí zpracovat replika master. Multi master model proto umožňuje provedení aktualizace dat na jakékoliv replice spravující příslušná data. Neexistuje zde žádná replika v roli master nebo slave, ale v každém klastru vystupuje uzel v roli koordinátor (coordinator), který požadavky šíří rovnoměrně mezi repliky. Pokud koordinátor zaznamená výpadek, jeho roli převezme následující uzel po směru hodinových ručiček. Multi master model nabízí několik mechanismů jak docílit, aby všechny repliky vlastnící aktualizovaná data byly ve stejném stavu.

3.9.1 Quorum dvoufázový commit

Jako u relačních databází [10] i NO-SQL databáze nabízí dvoufázový commit pro zajištění striktní konzistence na každé replice. Jako u relačních databází probíhá commit ve dvou fázích:

- Prepafe phase,
- commit phase.

Prepare phase - ve fázi přípravy koordinátor zašle požadavek každé replice, aby potvrdila, že je připravena provést aktualizaci dat. Každá replika pak musí zapsat data do logu a pokud je vše v pořádku odpoví pozitivně koordinátorovi.

Commit phase - koordinátor obdrží pozitivní odpověď od všech replik a začíná fáze commit. Nyní odešle všem replikám požadavek na commit a každá replika zapíše do svého log souboru informaci o potvrzení aktualizace. Jakmile podruhé všechny repliky odpoví pozitivně, aktualizace je považována za kompletní.

Dvou fázový commit snižuje propustnost databáze, protože obě fáze probíhají synchronně. V případě, že by data spravovalo mnoho replik, čas na provedení dvou fazového commitu může významně růst [9]. Dalším problém je, že pokud alespoň jedna neodpoví pozitivně v jedné z fází, operace je automaticky požadována za nedokončenou a aktualizace je ztracena.

U NO-SQL databází byl zaveden tzv. Quorum dvoufázový commit. Jinými slovy commit založený na souhlasu většiny. Proto jsou zavedeny parametry **R**, **W** a **N**. Kde parametr R znamená počet replik, které musí odpovědět pozitivně na operací čtení, W je počet replik, které musí pozitivně odpovědět na operací zápis a N celkový počet replik. Pokud tedy na požadavek operace zápis odpoví, alespoň W replik, operace je provedena

bez ohledu na ostatní repliky, které mohou být v tu chvíli nedostupné. Tento postup je mnohem efektivnější z pohledu pravděpodobnosti výpadku jednotlivých replik [9].

Podobně pak při zpracování operace čtení musí odpovědět pozitivně R replik. Data jsou tedy vráceny ze všech R replik ke koordinátorovi a ten vrátí nejaktuálnější verzi podle časové značky, protože všechny repliky nemusely zpracovat poslední aktualizaci dat.

Parametry R a W jsou nastaveny na straně klienta při odesílání příslušného požadavku, tedy R při operaci čtení a W při operaci zápis, v závislosti na tom, jako úroveň konzistence klient požaduje. Pro zachování konzistence založené na modelu quorum musí platit [9]:

$$W + R \quad > \quad N$$

Pro dodržení striktní konzistence platí W = N a R = 1. Pokud klient toleruje nižší úroveň konzistence avšak stále založené na modelu quorum pak stačí, aby parametry R a W byly nastaveny, aby platilo:

$$W + R \quad <= \quad N$$

Může nastat situace, kdy klient nevyžaduje striktní konzistenci a pak není nutné třeba používat quorum commit. V takové situaci se používá gossip model, ve kterém jsou aktualizace dat šířeny asynchroně pomocí gossip zpráv, tak že každá replika dojde ke konzistentnímu stavu eventuálně.

3.9.2 Zpracování operací delete

V rámci multi-master modelu je nutné pečlivě zpracovat operaci delete. Může se totiž stát, že ne všechny repliky zpracují operaci delete a operace tedy selže. Proto jsou příslušná data nejdříve označena jako smazaná, ale stále jsou uchována. Jakmile všechny repliky oznámí, že operaci delete dokončily, pak jsou příslušná data smazána.

3.10 Vektorové hodiny

Vektorové hodiny (vector clock) se používají k řešení konfliktů při souběžném zápisu na více replikách. Vektorové hodiny jsou definovány jako n-tice V[0], V[1],.., V[n] hodinových hodnot každé repliky. Každá replika uchovává n-tici hodnot, které reprezentují stav příslušné repliky a všech replik, které jsou k příslušné replice přidružené (spravují stejná data). Stav je zde chápán jako aktuální verze dat. Taková n-tice by pak mohla vypadat takto $V_i[0]$ pro první repliku, $V_i[i]$ pro repliku samotnou a $V_i[n]$ pro poslední repliku. Hodnota vektorových hodin může být časová značka, číslo verze nebo jiná ordinální hodnota.

Vektorové hodiny jsou aktualizovaný v případě:

- Po vykonání interní operace příslušná replika zvýší své vektorové hodiny $V_i[i]$. Výsledek interní operace je tedy okamžitě znám příslušné replice.
- Pokud replika i chce poslat zprávu replice k, nejprve zvýší hodnotu svých vektorových hodin $V_i[i]$ a přiloží ji do zprávy spolu se stavem ostatních replik. Replika k tak zjistí interní stav repliky i a také stav ostatních replik známý replice i.
- Jakmile replika i obdrží zprávu od repliky j, nejprvé zvýší hodnotu svých vektorových hodin $V_i[i]$ a poté provede synchronizaci s vektorovými hodinami ve zprávě $V_{message}$. Pro výsledek synchronizace platí:

  $$V_i \quad = \quad max(V_i, V_{message})$$

  Při porovnání dvou vektorových hodin V_i a V_j ke zjištění dílčího uspořádání se používá následující pravidlo:

  $$V_i \; > \; V_j, \; pokud \, \forall k \; V_i[k] \; > \; V_j[k]$$

  Jestliže neplatí $V_i > V_j$ ani $V_i < V_j$, tak nelze konflikt způsobený souběžným zápisem vyřešit a je nutné, aby byl vyřešen na straně klienta [5] .

Vektorové hodiny mohou být také použity k propagaci změn v rámci modelů state transfer a operation transfer.

3.10.1 State transfer model s využitím vektorových hodin

Stejně jako v kapitole 3.8 i zde si jednotlivé repliky uchovávají verzovací stromy, ale navíc také vektorové hodiny. Každý klient si pak uchovává vektorové hodiny pro data, která četl nebo aktualizoval. Při čtení dat nejprve klient odešle své vektorové hodiny (pokud je má). Replika poté odpoví verzí dat, která předchází vektorovým hodinám odeslaných klientem (tímto je zaručen monotonic read consistency) a svými vektorovými hodinami. Klient teď synchronizuje své vektorové hodiny s hodinami, které obdržel v odpovědi, zvýší své vektorové hodiny a musí řešit potencionální konflikty. Kdyby konflikty v tuto chvíli nevyřešil mohlo by se stát, že bude pracovat s verzí dat, které již nejsou aktuální vzhledem k verzi, keré udržuje příslušná replika [5].

Při aktualizaci dat, stejně jako u čtení, musí klient přiložit k požadavku na aktualizaci své vektorové hodiny. Replika porovná své vektorové hodiny a pokud je hodnota vektorových hodin klienta nižší než aktuální stav repliky požadavek na aktualizaci zahodí. V opačném případě replika aktualizaci dat provede. Repliky zoodpovědné za stejná data si tedy v pozadí pomocí gossip protokolu vyměňují své vektorové hodiny a snaží se je spojovat tak, aby byly všechny repliky synchronizovány [5].

3.10.2 Operation state model s využitím vektorových hodin

Tak jako u operation state modelu popsaného v kapitole 3.8 je nutné dodržet správné pořadí operací. V tomto modelu jsou k tomu využity vektorové hodiny místo časových značek. Každá replika v tomto modelu udržuje následující:

- V_{state}: vektorové hodiny odpovídající stavu po poslední aktualizaci dat.
- V_i: vektorové hodiny, které v porovnání s V_{state}, jsou přiloženy k operacím, které již byly provedeny.
- V_j: vektorové hodiny přijaté poslední gossip zprávou od repliky j (platí pro všechny ostatní repliky).

Při čtení klient přiloží své vektorové hodiny (pokud již četl konkrétní data). Replika vrátí poslední verzí dat, jinými slovy buď verzi, která odpovídá verzi vektorových hodin klienta, nebo logicky následující verzi podle vektorových hodin repliky samotné[5].

Při aktualizaci dat replika uchová operaci zápisu ve svém bufferu, dokud ji není možné provést v závislosti na svém stavu, kdy bere v potaz vektorové své hodiny (V_{state}, V_i, V_j:) a frontu již provedených operací. Operaci uložené do bufferu jsou přiděleny dvoje vektorové hodiny: V_{client} reprezentující stav dat, která zná klient v okamžiku potvrzení operace a V_{client} reprezentující stav repliky, na které se provádí aktualizace. Jakmile všechny operace, které logicky předchází přijaté operaci byly provedeny, jsou provedeny, pak může být provedena aktualizace obdržená od klienta[5].

V tomto modelu si repliky vyměňují frontu operací, které musí být provedeny. Po každé výměně se repliky musí rozhodnout, které operace mohou být provedeny a v jakém pořadí. Pokud může být v jedné chvíli provedeno více operací replika určí pořadí podle vektorových hodin, které jsou přiloženy ke každé operaci a operace pak ve správném pořadí provede. Problém však nastává v případě, kdy dojde k souběžným zápisům na více replikách pak se musí operace rozdělit do dvou skupin:

- Souběžné aktualizace jsou komutativní, na pořadí tedy nezáleží a mohou být provedeny v libovolném pořadí.
- Souběžné aktualizace nejsou komutativní, na pořadí záleží a nelze jej určit porovnáním vektorových hodin. V takovém případě je zaveden tzv. globální čítač, který přiřazuje logické operací pro všechny požadavky. Jednotlivé operace se pak provádí v pořadí určeném tímto čítačem [5].

Po provedení operace(í) na replice může být každá operace odebrána z bufferu až poté, co odpoví všechny repliky příslušných dat, že také provedly tuto operaci.

4 Rešerše databázových systémů podporující různou konzistenci operací

Tato kapitola poskytuje základní přehled NO-SQL databází a popisuje datový model jednotlivých databází a možnosti nastavení úrovně konzistence jednotlivých operací.

4.1 Amazon DynamoDB

DynamoDB se nedá chápat jako databáze jako taková. Jedná se o webovou službu poskytovanou společností Amazon. Skutečnost, že se jedná webovou službu poskytuje řadu výhod. První výhodou je, že není potřeba jakékoliv administrace databáze, databázového serveru. Není tedy nutné se starat o aktualizace databáze, či operačního systému, na kterém databáze běží a ani o hardware samotný. Dále DynamoDB automaticky alokuje další úložiště jakmile objem uložených dat narůstá, celá webová služba totiž běží v prostředí Amazon Elastic Compute Cloud (Amazon EC2). Amazon také garantuje průměrné odezvy na straně serveru a to v řádu desítek milisekund [11]. Správa databáze z pohledu jejího schématu pak probíhá skrz jednoduché a intuitivní webové rozhraní AWS Management Console [14]. Přistupovat k DynamoDB lze pomocí klientů vytvořených v programovacích jazycích Java, JavaScript, PHP, Ruby, Python a .NET [12].

4.1.1 Datový model DynamoDB

Oproti jiným NO-SQL databázím nabízí DynamoDB datový model vzdáleně podobný relačním databázím. Datový model se zde skládá z tabulek, položek a atributů. Databáze je kolekcí tabulek a jednotlivé tabulky jsou kolekcí položek a každá položka je kolekcí atributů. Oproti relačním databázím však každá položka v tabulce nemá určený počet atributů. Jediný povinný atribut je primární klíč [13] a jedná se tedy o tzv. key-value store (kapitola 3.1).

4.1.2 Nastavení konzistence operací

DynamoDB nabízí dvě možnosti nastavení úrovně konzistence pro operaci čtení: eventuálně konzistentní čtení a konzistentní čtení.

4.1.2.1 Eventuálně konzistentní čtení Při tomto nastavení operace čtení se může stát, že klient neobdrží poslední verzi zapsaných dat, protože poslední verze ještě nebyla kompletně replikována. DynamoDB však zaručuje replikaci během vteřiny [15].

4.1.2.2 Konzistentní čtení Při nastavení konzistentního čtení DynamoDB vrátí "nejčerstvější data", tedy verzi, u které byla úspěšně provedena poslední aktualizace. Konzistentní čtení však může snižovat dostupnost dat v případě síťových výpadků [15]. Nastavení konzistence pro operace zápisu není u DynamoDB možné. Důležitou vlastnostní nastavení konzistence operací je, že je možné pro každou operaci tuto úroveň nastavit

jinak a to i v rámci jedné aplikace. Je tedy možné rozhodnout, pro které operace čtení je potřeba nastavit úroveň konzistentní čtení a pro eventuálně konzistentní čtení.

4.2 Cassandra

NO-SQL databáze Cassandra je navržena jako řešení pro aplikace, které vyžadují vysokou škálovatelnost, výkon, flexibilní design a nepřetržitou dostupnost. K zvýšení dostupnosti nabízí Cassandra flexibilní replikaci dat mezi datovými centry či v cloudu [16]. Data uložena v databázi Cassandra mohou být zapisovány i čteny na uzlu, který může ležet teoreticky kdekoliv na světě. Cassandra zaručuje trvalost dat jako při použití principu ACID (2.4.1). K tomu používá tzv. commit log, který uchovává informace o všech operacích zápis a to i redundantně právě ke zvýšení trvalosti dat v případě hardwarových selhání.

Pro přístup k databázi Cassandra je možné využít široké spektrum klientů v programovacích jazycích Python, Java, Ruby a PHP [20]. Databázi Cassandra je možné spustit na operačních systémech Windows, Linux i MacOS. Ke stažení je pak na stránkách společnosti Datastax [23]. Součástí instalace databáze Cassandra je také CQL konzole, přes kterou je možné spouštět dotazy na databázi. Jazyk CQL (Cassandra Query Language) je velmi podobný jazyku SQL.

4.2.1 Datový model

Datový model databáze Cassandra je navržen pro distribuci dat ve velkém měřítku. Narozdíl od relačních databází nevyužívá ACID, aby zvýšil výkon a dostupnost databáze [21]. Jedná se o key-value datový model a Skládá se z keyspace a columnfamily.

Keyspace - klíčový prostor, se dá přirovnat k databázi u tradičních relačních databází, který se skládá z jednotlivých columnfamilies. Konfigurace klíčového prostoru vyjadřuje konfiguraci jednotlivých columnfamilies, které se v příslušném klíčovém prostoru nachází [21].

Columnfamily - kolekce jednotlivých řádků, připomínající tabulku u relačních databází. Každý řádek je identifikován svým klíčem (key). Jednotlivé řádky v rámci jedné columnfamily nemusí mít stejný počet atributů. Jednotlivé atributy také není nutné předem definovat. Definice je možná v podstatě ať už při vytváření columnfamily, nebo při plném běhu databáze [21].

4.2.2 Nastavení konzistence operací

Databáze Cassandra nabízí možnost nastavení konzistence jak pro operaci čtení tak pro operaci zápis. Každá aplikace tedy může využít jiné úrovně konzistence v závislosti na případu užití.

4.2.2.1 Nastavení konzistence pro operace čtení

Při čtení do databáze Cassandra úroveň konzistence vyjadřuje počet replik, které musí odpovědět než bude odpověď

odeslání klientovi. Možnosti nastavení úrovně konzistence pro operaci čtení zobrazuje tabulka 3. Jednotlivé úrovně jsou seřazeny od nejslabší po nejsilnější úroveň.

Úroveň konzistence	Popis
ONE	Právě jedna replika musí odpovědět, aby byla operace považována za úspěšnou a data byla vrácena klientovi.
QUORUM	Budou dotazány všechny repliky a vrácena budou data, která mají nejnovější časovou značku na alespoň (N/2)+1 (většině) replikách (N vyjadřuje počet replik).
LOCAL QUORUM	Budou dotazány všechny repliky v rámci datacentra a vrácena budou data s nejnovější časovou značku na většině replik.
EACH QUORUM	Budou dotazány všechny repliky v každém datacentru a vrácena budou data s nejnovější časovou značku na většině replik.
ALL	Dotazány budou všechny repliky a data budou vrácena klientovi jakmile všechny repliky odpoví. Jakmile alespoň jedna replika neodpoví, operace selže.

Tabulka 3: Možností nastavení úrovně konzistence operace čtení databáze Cassandra [22]

4.2.2.2 Nastavení konzistence pro operace zápis

Při zápisu do databáze Cassandra úroveň konzistence vyjadřuje počet replik, na kterých musí být zápis úspěšný než se vrátí potvrzení úspěšnosti operace klientovi. Jednotlivé úrovně jsou seřazeny od nejslabší po nejsilnější úroveň v tabulce 4.

Úroveň konzistence	Popis
ANY	Zápis musí být proveden na alespoň jedné replice. V případě, že všechny repliky zoodpovědné za příslušný klíč nejsou funkční, zaznamená se zápis na jiném uzlu v klastru. V tomto případě však nebude možné tato data číst do doby než budou data replikována příslušným replikám.
ONE	Zápis musí být proveden na alespoň jedné replice.
QUORUM	Zápis musí být proveden na většině replik ((N/2)+1).
LOCAL QUORUM	Zápis musí být proveden na většině replik v rámci datacentra.
EACH QUORUM	Zápis musí být proveden na většině replik v rámci všech datacenter.
ALL	Zápis musí být proveden na všech replikách zoodpovědných za příslušný klíč.

Tabulka 4: Možností nastavení úrovně konzistence operace zápis databáze Cassandra [22]

Jako u DynamoDB je možné nastavit úroveň konzistence pro každou operaci zvlášť.

4.3 Riak

Riak je open source distribuovaná databáze navržená především pro vysokou dostupnost, odolnost vůči výpadkům a vysokou škálovatelnost [25]. Pro připojení k databázi Riak jsou dostupní klienti v programovacích jazycích Erlang, Java, PHP, Python, Ruby, C/C++ a JavaScript [26]. Databázi Riak je možné instalovat a spustit na operačních systémech Linux a MacOS a je dostupná ke stažení na [27]

4.3.1 Datový model

Riak používá jednoduchý key-value datový model. Data jsou uchovávána ve struktuře zvané bucket. Bucket představuje tabulku jako u relačních databází. Každý řádek pak obsahuje unikátní identifikátor (key) a hodnotu, nebo množinu hodnot.

4.3.2 Nastavení konzistence operací

Pro nastavení konzistence používá Riak parametry *W* a *R*. Parametr W vyjadřuje kolik replik musí provést zápis, aby byla operace považována za úspěšnou a parametr R vyjadřuje kolik replik musí odpověď na operaci čtení předtím, než jsou data vrácena klientovi.

Riak zavádí symbolické označení úrovní konzistence jako v tabulce 5. Toto označení je platné jak pro operace čtení tak zápis.

Úroveň konzistence	Popis
ALL	Všechny repliky musí odpovědět. Parametry W a R jsou rovny replikačnímu faktoru.
ONE	Alespoň replika musí odpovědět. Parametry W a R jsou pak nastaveny na hodnotu 1.
QUORUM	Většina replik musí odpovědět ((N/2)+1). Při nastavení replikačního faktoru na 3 jsou pak parametry W a R nastaveny na hodnotu 2.
DEFAULT	Při použití této úrovně je použito libovolna úroveň konzistence, pak jsou nastaveny parametry W a R na libovolnou hodnotu.

Tabulka 5: Možností nastavení úrovně konzistence databáze Riak

I databáze Riak umožňuje nastavení úroveně konzistence pro každou operaci zvlášť.

5 Testování propustnosti NO-SQL databází

Pro testování propustnosti byly po dohodě s vedoucím práce vybrány databáze Cassandra (použita edice Datastax community 1.0.0) a Riak (použita verze 1.2.1). Výběr byl ovlivněn především možnostmi nastavení konzistence operací. Pro otestování propustnosti byl pro každou databázi vytvořen klastr čítající 5 uzlů. Jednotlivé uzly byly vytvořeny jako virtuální stroje na serveru dbedu.cs.vsb.cz s hardwarovou konfigurací : jedno-jádrový virtuální procesor a 1.5 GB operační paměti v prostředí Hyper-V. Připojení k serveru dbedu bylo realizováno pomocí VPN. Pro databázi Riak byl zvolen operační systém Linux Ubuntu 13.04 Raring Ringtail dostupný na [18] a pro databázi Cassandra Windows 2008 R2 server dostupný pro studenty na [19]. Replication factor byl nastaven na hodnotu 3. Pět různých vytížení pro jednotlivé úrovně konzistence bylo spuštěno zároveň v pěti vláknech s nastaveným poměrem operací čtení a zápis jak znázorňuje tabulka 7. Každé vlákno pak spustí sto tisíc operací v náhodném pořadí podle tohoto poměru a objem přenesených objema směry dat se dá odhadnout kolem 100MB. Vytížení jak pro databázi Cassandra tak Riak je vytvořeno v programovacím jazyce Python. Databáze byly předem naplněny jedním miliónem záznamů (každý záznam zabírá přibližně 20 KB, proto není zahrnuta operace insert. Každý výsledek vytížení je pak uložen v souboru s příponou txt. Všechny výsledky se nachází v A.

Jednotlivé zatížení databáze měří propustnost databáze tak, že měří dobu odezvy provedení operace zápisu a čtení. U databáze Cassandra je toto měření provedeno jak na straně klienta tak na straně serveru. U databáze Riak, nebylo možné provést měření na straně serveru, protože databáze samotná toto měření nepodporuje.

	UPDATE	**SELECT**	**DELETE**
Vytížení 1	35%	55%	10%
Vytížení 2	55%	35%	10%
Vytížení 3	35%	40%	25%
Vytížení 4	33%	33%	24%
Vytížení 5	40%	25%	35%

Tabulka 6: Poměr operací pro testování propustnosti

5.1 Cassandra

Pro testování propustnosti byl vybrán klient Pycassa volně dostupný na [17].

5.1.1 Inicializace klastru

Pro otestování propustnosti databáze Cassandra je potřeba nastavit správně jednotlivé uzly tak, aby tvořily klastr. Pro správnou funkci je nutné nastavit parametry: *cluster_name, initial_token, seed_provider, listen_adress, rpc_address a endpoint_snitch*. Tyto konfigurační se nachází v souboru *cassandra.yaml*, který se nachází v instalačním adresáři.

Podrobný návod k nastavení klastru databáze Cassandra se nachází na [30], jedná se o návod pro operační Linux, ale lze podle něj postupovat i při konfiguraci na operačním systému Windows. Pouze spuštění a zastavení Cassandry probíhá přes správce úloh. Návod, jenž byl použit při spuštění klastru, již není publikován.

5.1.2 Měření na straně klienta

Tabulka 7 obsahuje počet operací provedených jednotlovými vytíženími, tabulka 8 průměrnou odezvou operací čtení a zápis na straně klienta a tabulka 9 obsahuje trvání jednotlivých vytížení.

	ONE	QUORUM	ALL
Vytížení 1	277140	277456	277641
Vytížení 2	178695	178583	177407
Vytížení 3	202635	203215	203364
Vytížení 4	167999	168276	167892
Vytížení 5	128205	127052	129227

	ANY	ONE	QUORUM	ALL
Vytížení 1	222111	222855	222539	222359
Vytížení 2	321683	321304	321412	322593
Vytížení 3	297254	297363	296784	296618
Vytížení 4	331911	332000	331724	330652
Vytížení 5	370702	371793	371048	370773

Tabulka 7: Počet operací čtení a zápis Cassandra na straně klienta

	ONE	QUORUM	ALL
Vytížení 1	0.0057	0.0066	0.0066
Vytížení 2	0.0055	0.0059	0.0064
Vytížení 3	**0.0072**	0.0058	0.0059
Vytížení 4	0.0054	0.0064	0.0065
Vytížení 5	0.0059	**0.0183**	0.0062

	ANY	ONE	QUORUM	ALL
Vytížení 1	0.0051	0.0053	0.0058	0.0063
Vytížení 2	0.0053	0.0052	0.0055	0.0061
Vytížení 3	0.0053	**0.0070**	0.0053	0.0056
Vytížení 4	0.0052	0.0051	0.0056	0.0058
Vytížení 5	0.0052	0.0054	0.0063	0.0058

Tabulka 8: Průměrná odezva operace čtení a zápis Cassandra na straně klienta[s]

	ANY	ONE	QUORUM	ALL
Vytížení 1	49	50	57	63
Vytížení 2	51	48	52	59
Vytížení 3	58	73	57	59
Vytížení 4	60	58	65	67
Vytížení 5	61	62	126	66

Tabulka 9: Celková doba vytížení [minuty]

Při měření odezvy jednotlivých operací se vyskytly odchylky a jsou označeny tučně v tabulce 8. I přes tyto odchylky lze vidět, že při nastavení silnější konzistence se odezva zvyšuje. Což je předvídatelné - na operaci musí odpovědět více replik. Např. rozdíl mezi úrovní ANY a ALL je přibližně 20%.

5.1.3 Měření na straně serveru

Na straně serveru probíhalo měření v prostředí Datastax OpsCenter [29], které monitoruje databázi Cassandra. Jednotlivé výsledky jsou průměrem pro jednotlivá vytížení -

výsledky jsou odečteny z grafů v prostředí OpsCenter. Tyto grafy se nachází v přílohách A. Měřeny byly počty operací čtení a zápis za sekundu a průměrná odezva obou operací. Tabulky 10 a 11 obsahují počty jednotlivých operací za sekundu a průměrnou odezvu těchto operací v sekundách.

	ONE	QUORUM	ALL
Vytížení 1	60	79	68
Vytížení 2	47	86	61
Vytížení 3	59	53	43
Vytížení 4	34	46	32
Vytížení 5	36	**9**	43

	ANY	ONE	QUORUM	ALL
Vytížení 1	76	132	85	75
Vytížení 2	95	109	110	69
Vytížení 3	93	138	124	91
Vytížení 4	101	108	161	96
Vytížení 5	101	171	59	93

Tabulka 10: Počet operací čtení a zápis Cassandra na straně serveru [s]

	ONE	QUORUM	ALL
Vytížení 1	0.0031	0.0022	0.0067
Vytížení 2	0.0029	**0.11**	0.0067
Vytížení 3	0.0034	**0.61**	0.0055
Vytížení 4	0.0035	46	**0.068**
Vytížení 5	**0.0058**	**3.01**	0.0060

	ANY	ONE	QUORUM	ALL
Vytížení 1	0.0009	0.0022	0.0035	0.0038
Vytížení 2	**0.0011**	0.0022	**0.0070**	**0.32**
Vytížení 3	**0.00069**	0.0020	0.0041	0.0035
Vytížení 4	**0.00064**	0.0022	0.0025	0.0044
Vytížení 5	**0.00062**	0.0022	**3.01**	0.0036

Tabulka 11: Průměrná odezva operací čtení a zápis Cassandra na straně serveru[s]

V tabulce 11 jsou tučně vyznačeny odchylky v měření. Ty mohly být způsobeny vytížením fyzického serveru dbedu, na kterém běžely jednotlivé uzly, nebo dokonce výpadkem některého z uzlů. I zde je však vidět, že s vyšší úrovní konzistence se odezva zvyšuje. Při porovnání úrovně konzistence ANY a ALL je rozdíl dokonce 400%.

5.2 Riak

Pro testování propustnosti databáze Riak byl vybrán klient riak-python-client volně dostupný na [28].

5.2.1 Inicializace klastru

Pro nastavení uzlů tak, aby tvořily klastr je potřeba nastavit správně následující parametry: *http, pb_ip a name*. První dva se nachází v souboru *app.config* a třetí v souboru *vm.args*. Oba soubory se nachází v adresáři etc v kořenovém adresáři instalace Riaku. Příklad takového nastavení:

Podrobný návod se nachází na [31].

5.2.2 Měření na straně klienta

Tabulka 12 obsahuje počet jednotlivých operací provedených jednotlivými vytíženími, tabulka 13 obsahuje odezvy jednotlivých operací v sekundách a tabulka 14 obsahuje dobu jednotlivých vytížení v minutách.

	ONE	QUORUM	ALL
Vytížení 1	276993	276880	277465
Vytížení 2	177947	177879	178115
Vytížení 3	202571	203432	202506
Vytížení 4	168359	277897	168812
Vytížení 5	128560	277352	128646

	ONE	QUORUM	ALL
Vytížení 1	223007	223120	222535
Vytížení 2	322053	322121	321885
Vytížení 3	297429	296568	297489
Vytížení 4	331641	222103	331173
Vytížení 5	371437	222648	371354

Tabulka 12: Počet operací čtení a zápis Riak na straně klienta

	ONE	QUORUM	ALL
Vytížení 1	0.0062	0.0060	0.0058
Vytížení 2	0.0059	0.0061	0.0058
Vytížení 3	0.0059	0.0061	0.0060
Vytížení 4	0.0061	0.0058	0.0057
Vytížení 5	0.0060	0.0058	0.0057

	ONE	QUORUM	ALL
Vytížení 1	0.0067	0.0065	0.0065
Vytížení 2	0.0065	0.0061	0.0065
Vytížení 3	0.0063	0.0067	0.0066
Vytížení 4	0.0065	0.0062	0.0064
Vytížení 5	0.0065	0.0061	0.0063

Tabulka 13: Průměrná odezva operací čtení a zápis Riak na straně klienta[s]

	ONE	QUORUM	ALL
Vytížení 1	84	82	79
Vytížení 2	92	96	90
Vytížení 3	97	101	99
Vytížení 4	108	77	106
Vytížení 5	89	76	107

Tabulka 14: Celková doba vytížení [minuty]

U databáze je z výsledků vidět, že sice nebyly zaznamenány žádné větší výkyvy v měření, ale zde se již neprojevuje tendence růstu odezvy v závislosti na zvyšování úrovně konzistence. Měření je tedy zřejmě ovlivněno vnějšími vlivy jako je síťová odezva nebo zatížení fyzického serveru.

6 Testování propustnosti s vyšší hardwarovou konfigurací

Po prozkoumání výsledků prvotního testování propustnosti bylo společně s vedoucím práce rozhodnuto, že by bylo dobré propustnost databází otestovat na uzlech s vyšší hardwarovou konfigurací. U některých měření byly totiž zaznamenány odchylky jak u databáze Cassandra (vyznačeny tučně v tabulkách 8 a 11) tak u databáze Riak. Proto byla na každém uzlu zvýšena operační paměť zvýšena z původní hodnoty 1.5 GB na 4 GB pro minimalizaci diskových operacích na jednotlivých uzlech.

6.1 Cassandra

6.1.1 Měření na straně klienta

Tabulka 15 obsahuje počet operací provedených jednotlovými vytíženími, tabulka 16 průměrnou odezvou operací čtení a zápis na straně klienta a tabulka 17 obsahuje trvání jednotlivých vytížení.

	ONE	QUORUM	ALL
Vytížení 1	274766	277075	277785
Vytížení 2	177787	178323	178325
Vytížení 3	202285	202952	203532
Vytížení 4	167626	168392	168190
Vytížení 5	128977	128814	128698

	ANY	ONE	QUORUM	ALL
Vytížení 1	222735	223078	222920	222215
Vytížení 2	321450	322213	321672	321649
Vytížení 3	296447	297715	297048	296468
Vytížení 4	331218	332374	331607	331808
Vytížení 5	371001	371023	371175	371302

Tabulka 15: Počet operací čtení a zápis Cassandra na straně klienta

	ONE	QUORUM	ALL
Vytížení 1	0.0056	0.0069	0.0064
Vytížení 2	0.0061	0.0063	0.0070
Vytížení 3	0.0064	0.0062	0.0065
Vytížení 4	0.0061	0.0058	0.0064
Vytížení 5	0.0056	0.0055	0.0059

	ANY	ONE	QUORUM	ALL
Vytížení 1	0.0054	0.0053	0.0058	0.0061
Vytížení 2	0.0053	0.0056	0.0059	0.0067
Vytížení 3	0.0054	0.0057	0.0059	0.0062
Vytížení 4	0.0054	0.0056	0.0052	0.0060
Vytížení 5	0.0050	0.0053	0.0051	0.0056

Tabulka 16: Průměrná odezva operace čtení a zápis Cassandra na straně klienta [s]

	ANY	ONE	QUORUM	ALL
Vytížení 1	53	50	58	57
Vytížení 2	51	53	56	63
Vytížení 3	60	62	63	66
Vytížení 4	64	64	60	68
Vytížení 5	58	60	59	63

Tabulka 17: Celková doba vytížení [minuty]

Z výsledků je patrné, že zvýšení hardwarové konfigurace serveru pro databázi Cassandra elimonovalo odchylky na straně klienta a i zde je vidět tendence růstu odezvy se zvyšující se úrovní konzistence a stejně jako při testování s nižší hardwarovou konfigurací přibližně o 20%.

6.1.2 Měření na straně serveru

Tabulka 18 obsahuje počet jednotlivých operací za sekundu na straně serveru s vyšší hardwarovou konfigurací. Tabulka 19 pak dobu odezvy jednotlivých operací v sekundách.

	ONE	QUORUM	ALL
Vytížení 1	20	86	56
Vytížení 2	40	54	81
Vytížení 3	60	53	119
Vytížení 4	51	34	106
Vytížení 5	92	55	120

	ANY	ONE	QUORUM	ALL
Vytížení 1	73	105	101	61
Vytížení 2	107	133	140	81
Vytížení 3	98	118	141	119
Vytížení 4	126	118	103	106
Vytížení 5	115	129	140	120

Tabulka 18: Počet operací čtení a zápis Cassandra [s]

	ONE	QUORUM	ALL
Vytížení 1	0.0037	**0.36**	0.0069
Vytížení 2	0.0058	0.0082	**0.0085**
Vytížení 3	**0.0081**	0.0072	0.0072
Vytížení 4	**0.0087**	0.0045	0.0071
Vytížení 5	0.0052	**0.0100**	0.0055

	ANY	ONE	QUORUM	ALL
Vytížení 1	0.0015	0.0026	0.0021	0.0035
Vytížení 2	0.0011	0.0029	0.0040	0.0034
Vytížení 3	0.0011	**0.0050**	0.0034	0.0027
Vytížení 4	0.0013	**0.0046**	0.0011	0.0026
Vytížení 5	0.0021	**0.0052**	**0.01**	**0.0055**

Tabulka 19: Průměrná odezva operací čtení a zápis Cassandra [s]

V tabulce 19 jsou vyznačeny odchylky, které se objevily i přes zvýšení hardwarové konfigurace uzlů. I v tomto případě byly s největší pravděpodobností způsobeny vytížením fyzického serveru dbedu nebo výpadkem některého z uzlů. Ve vytíženích, které neobsahují odchylky lze opět vidět tendenci růstu odezvy v závislosti na zvyšovaní úrovně konzistence. Po dohodě s vedoucím tak bylo rozhodnuto, že bude dobré otestovat propustnost databáze Cassandra i v případě výpadku jednoho z uzlů.

6.2 Riak

Tabulka 20 obsahuje počet jednotlivých operací provedených jednotlivými vytíženími, tabulka 13 obsahuje odezvy jednotlivých operací v sekundách a tabulka 22 obsahuje dobu jednotlivých vytížení v minutách.

	ONE	QUORUM	ALL
Vytížení 1	277150	277176	277584
Vytížení 2	178373	178454	178117
Vytížení 3	202268	202659	203442
Vytížení 4	168285	203142	168799
Vytížení 5	128694	128374	129132

	ONE	QUORUM	ALL
Vytížení 1	222850	222824	222416
Vytížení 2	321627	321546	321883
Vytížení 3	297732	297341	331201
Vytížení 4	331715	296858	331173
Vytížení 5	371306	371626	370868

Tabulka 20: Počet operací čtení a zápis Riak na straně klienta

Při testování propustnosti databáze Riak s vyšší hardwarovou konfigurací se objevily paradoxně odchylky v jednotlivých odezvách a jsou vyznačeny v tabulce . I u databáze

	ONE	QUORUM	ALL
Vytížení 1	**0.0142**	0.0058	0.0059
Vytížení 2	0.0077	0.0059	0.0060
Vytížení 3	0.0058	0.0058	0.0059
Vytížení 4	0.0059	0.0058	0.0058
Vytížení 5	0.0061	0.0065	0.0058

	ONE	QUORUM	ALL
Vytížení 1	**0.0150**	0.0062	0.0058
Vytížení 2	**0.0150**	0.0063	0.0065
Vytížení 3	0.0061	0.0062	0.0062
Vytížení 4	0.0062	0.0063	0.0061
Vytížení 5	0.0065	0.0061	0.0061

Tabulka 21: Průměrná odezva operací čtení a zápis Riak na straně klienta[s]

	ONE	QUORUM	ALL
Vytížení 1	187	77	78
Vytížení 2	115	88	91
Vytížení 3	92	92	93
Vytížení 4	102	94	100
Vytížení 5	89	111	105

Tabulka 22: Celková doba vytížení (minuty)

Riak mohlo dojít k výpadku některého z uzlů a proto bylo dohodnuto, že výpadek uzlu bude simulován i pro databázi Riak.

7 Simulace řízeného výpadku uzlu

Testování řízeného probíhalo tak, že bylo spuštěno vytížení jako v předchozích testech, ale pouze pro 10000 operací a po provedení přibližně tisíce operací byl jeden uzel odpojen od sítě. U databáze Cassandra i Riak byl výpadek uzlu simulován pro úroveň konzistence QUORUM. Ve chvíli, kdy dojde k výpadku uzlu, je nutná redistribuce dat mezi uzly, které jsou za ně nyní zoodpovědné a je zde přepoklad, že odevza databáze se zvýšší. Hardwarová konfigurace uzlů byla stejná jako u testování s vyšší hardwarovou konfigurací (6).

7.1 Cassandra

7.1.1 Měření na straně klienta

Tabulka 23 obsahuje počty jednotlivých operací a odezvu jednotlivých operací v sekundách. Doba vytížení byla u všech testů stejná - 6 minut, proto není tabulka uvedena.

	Poč. op. čtení	**Prům. od. op. čtení [s]**	**Poč. op. zápis**	**Prům. od. op. zápis [s]**
Vytížení 1	27539	0.0069	22461	0.0062
Vytížení 2	17772	0.0070	32228	0.0034
Vytížení 3	20381	0.0067	29619	0.0059
Vytížení 4	16671	0.0067	33329	0.0060
Vytížení 5	12862	0.0074	37137	0.0063

Tabulka 23: Výsledky simulace výpadku uzlu Cassandra

V porovnání s předchozím testováním (tabulka 16) se odezvy na straně klienta zvýšila, což je očekávaný výsledek vzhledem k potřebě redistribuce dat mezi zoodpovědnými uzly.

7.1.2 Měření na straně serveru

Tabulka 24 obsahuje počty jednotlivých operací za sekundu a průměrnou odezvu operací za sekundu.

	Poč. op. čtení	**Prům. od. op. čtení [s]**	**Poč. op. zápis**	**Prům. od. op. zápis [s]**
Vytížení 1	44	0.0020	53	0.0042
Vytížení 2	38	0.1801	93	0.0040
Vytížení 3	46	0.1801	114	0.0041
Vytížení 4	46	0.0814	137	0.0040
Vytížení 5	41	1.0368	168	0.0046

Tabulka 24: Výsledky simulace výpadku uzlu Riak

V porovnání s předchozím testováním (tabulka 19) se odezvy na straně serveru také zvýšila, což je očekávaný výsledek vzhledem k potřebě redistribuce dat mezi zoodpovědnými uzly.

7.2 Riak

Tabulka 25 obsahuje počty jednotlivých operací a odezvu jednotlivých operací v sekundách. Doba vytížení byla u všech testů stejná - 10 minut, proto není tabulka uvedena.

7.2.1 Měření na straně klienta

	Poč. op. čtení	Prům. od. op. čtení [s]	Poč. op. zápis	Prům. od. op. zápis [s]
Vytížení 1	27768	0.0060	22232	0.0065
Vytížení 2	17758	0.0060	32242	0.0065
Vytížení 3	20433	0.0061	29567	0.0065
Vytížení 4	16925	0.0059	33075	0.0064
Vytížení 5	12757	0.0064	0.0064	0.0068

Tabulka 25: Průměrná odezva operací čtení a zápis Riak [s]

Oproti předchozímu testování (tabulka 21) zaznamenal Riak mírné zvýšení odezvy jednotlivých operací. Což je, stejně jako u databáze Cassandra, očekáváno.

8 Porovnání databází Cassandra a Riak z pohledu propustnosti

Při srovnání jednotlivých testů provedených kapitolách 5, 6 a 7 je zřejmé, že obě databáze dosáhly velmi podobných výsledků jak pro operace čtení tak zápis jak při běžném provozu tak při simulování výpadku uzlu. Porovnání z pohledu serveru nebylo možné. Avšak databáze Riak přece jenom dosáhla stabilnějších výsledků z pohledu odezvy operací, protože zde nebylo zaznamenáno tolik odchylek jako u databáze Cassandra. Databáze však nabízí výhodou v tom, že je možné sledovat klastr na straně serveru, to prozatím u databáze Riak možné není. Monitorování databáze Riak bude možno až v některé z dalších verzí.

9 Závěr

Cílem této diplomové práce bylo prozkoumat možnosti současných distribuovaných databází z pohledu nastavení úrovně konzistence jednotlivých operací. Dále pak vybrat dvě databáze a otestovat jejich propusnost v závislosti na nastavení úrovně konzistence jednotlivých operací a tyto databáze porovnat.

Při měření propustnosti byly brány v potaz průměrné odezvy operací čtení a zápis na straně klienta. U databáze Cassandra navíc počet operací čtení a zápis za sekundu a také průměrná odezva těchto operací. U obou databází bylo dosaženo podobných výsledků co se týče průměrné odezvy jednotlivých operací. Ve většině případů dosahovaly obě databáze lepší odezvy v případě operací čtení. Ve většině proto, že se v měření vyskytly odchylky. Při všech měřeních byla zaznamenána tendence růstu odezvy v závislosti na zvyšování úrovně konzistence.

Tyto odchylky byly způsobeny možným výpadkem jednoho či více uzlů v průběhu testování, síťovou odezvou a také vytížením fyzického serveru, na kterém byly databáze spuštěny. Měření odezvy na straně klienta také bylo ovlivněno odezvou VPN připojení, přes které byly jednotlivé spuštěny.

Protože databáze dosáhly podobných výsledků, nedá se jednoznačně říct, která z nich je výkonnější. Obě nabízí dostačující možnosti pro nastavení úrovně konzistence jednotlivých operací a to pro každou operaci zvlášť. Je tedy možné zvolit úroveň konzistence pro každou aplikaci podle požadavků konkrétní aplikace, nebo části aplikace.

Při vypracování této diplomové práce jsem se seznámil s technologií NO-SQL databází což označuji za největší osobní přínos této diplomové práce, protože za dobu mého studia jsem se s touto technologií hlouběji nesetkal.

Další rozvoj této diplomové práce by se dal směřovat k nasazení některé nebo obou databází k reálné aplikaci a otestování propustnosti při ostrém provozu. Nasazení by mohlo probíhat, buď jako nasazení pouze NO-SQL databáze k aplikaci, nebo v kombinaci s klasickou relační databází. Jako vhodný případ užití se jeví aplikace, která vyžaduje zpracování velkého množství operací za sekundu.

10 Reference

[1] Narasimhan Ramji , *Facebook-like scalability with NoSQL*. [online]. 2011, s. - [cit. 2013-03-30]. Dostupné z: http://www.virtusa.com/blog/index.php/2011/04/facebook-like-scalability-with-nosql/

[2] Microsoft Corporation, *Scaling Out SQL Server*. [online]. 2011, s. - [cit. 2013-03-30]. Dostupné z: http://msdn.microsoft.com/en-us/library/aa479364.aspx

[3] Oracle Store, *Oracle Store*. [online]. 2013, s. - [cit. 2013-03-30]. Dostupné z: https://shop.oracle.com/

[4] Oracle® Real Application Clusters Administration and Deployment Guide, *Oracle docs*. [online]. 2013, s. - [cit. 2013-03-30]. Dostupné z: http://docs.oracle.com/cd/B28359_01/rac.111/b28254/admcon.htm

[5] NoSQL Databases, *NoSQL Databases*. [online]. 2013, s. - [cit. 2013-03-30]. Dostupné z: http://www.christof-strauch.de/nosqldbs.pdf

[6] Peter Neubauer *Graph Databases, NOSQL and Neo4j*. [online]. 2010, s. - [cit. 2013-04-11]. Dostupné z: http://www.infoq.com/articles/graph-nosql-neo4j

[7] Brian Ritchie, *An Introduction to Document Databases*. [online]. 2010, s. - [cit. 2013-04-12]. Dostupné z: http://weblogs.asp.net/britchie/archive/2010/08/12/document-databases.aspx

[8] Radim Bača, *Nerelační distribuované databáze*. [online]. 2011, s. - [cit. 2013-04-12]. Dostupné z: http://db.cs.vsb.cz/Workshops/2011-12-20.pdf

[9] Ricky Ho *NOSQL Patterns*. [online]. 2009, s. - [cit. 2013-04-14]. Dostupné z: http://horicky.blogspot.cz/2009/11/nosql-patterns.html

[10] Oracle® Database Administrator's Guide *Two-Phase Commit Mechanism*. [online]. 2012, s. - [cit. 2013-04-19]. Dostupné z: http://docs.oracle.com/cd/B28359_01/server.111/b28310/ds_txns003.htm

[11] Amazon *Amazon DynamoDB Features and Benefits*. [online]. 2013, s. - [cit. 2013-04-24]. Dostupné z: http://aws.amazon.com/dynamodb/features/

[12] Amazon *Amazon AWS documentation*. [online]. 2013, s. - [cit. 2013-04-30]. Dostupné z: http://aws.amazon.com/documentation/

[13] Amazon *Dynamo Faqs*. [online]. 2013, s. - [cit. 2013-04-24]. Dostupné z: http://aws.amazon.com/dynamodb/faqs/#What_is_the_Data_Model

[14] Amazon *AWS Management Console*. [online]. 2013, s. - [cit. 2013-04-30]. Dostupné z: http://aws.amazon.com/console/

[15] Amazon *Amazon DynamoDB developer guide*. [online]. 2013, s. - [cit. 2013-04-24]. Dostupné z: http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/APISummary.html

[16] Datastax *Apache Cassandra*. [online]. 2013, s. - [cit. 2013-04-24]. Dostupné z: http://www.datastax.com/what-we-offer/products-services/datastax-enterprise/apache-cassandra

[17] Tyler Hobbs *Pycassa*. [online]. 2013, s. - [cit. 2013-04-25]. Dostupné z: https://github.com/pycassa/pycassa

[18] Ubuntu *Stáhnout ubuntu*. [online]. 2013, s. - [cit. 2013-04-25]. Dostupné z: http://www.ubuntu.cz/ziskejte/stahnout

[19] MSDN *MSDN academic alliance*. [online]. 2013, s. - [cit. 2013-04-25]. Dostupné z: https://elms.cs.vsb.cz/IUV.php

[20] Datastax *Client Libraries and CQL Drivers*. [online]. 2013, s. - [cit. 2013-04-30]. Dostupné z: http://www.datastax.com/download/clientdrivers

[21] Appache *Cassandra data model*. [online]. 2013, s. - [cit. 2013-04-25]. Dostupné z: http://wiki.apache.org/cassandra/DataModel

[22] Datastax *Apache Cassandra 1.0 Documentation*. [online]. 2013, s. - [cit. 2013-04-25]. Dostupné z: http://www.datastax.com/docs/1.0/dml/data_consistency

[23] Datastax *Cassandra download*. [online]. 2013, s. - [cit. 2013-04-25]. Dostupné z: http://www.datastax.com/download/

[24] Datastax *Apache Cassandra 1.1 Documentation*. [online]. 2013, s. - [cit. 2013-04-30]. Dostupné z: http://www.datastax.com/docs/1.1/dml/using_cql

[25] Riak *Riak features*. [online]. 2013, s. - [cit. 2013-04-25]. Dostupné z: http://basho.com/riak/

[26] Riak *Client libraries*. [online]. 2013, s. - [cit. 2013-04-25]. Dostupné z: http://docs.basho.com/riak/1.1.4/references/Client-Libraries/

[27] Riak *Download Riak*. [online]. 2013, s. - [cit. 2013-04-30]. Dostupné z: http://docs.basho.com/riak/1.1.4/downloads/

[28] Brett Hazen *Riak python client*. [online]. 2013, s. - [cit. 2013-05-4]. https://github.com/basho/riakpythonclient

[29] Datastax *Datastax OpsCenter*. [online]. 2013, s. - [cit. 2013-04-25]. http://www.datastax.com/what-we-offer/products-services/datastax-opscenter

[30] Datastax *Configuring and Starting a Cassandra Cluster*. [online]. 2013, s. - [cit. 2013-05-4]. http://www.datastax.com/docs/0.8/install/cluster_init

[31] Riak *Basic Cluster Setup*. [online]. 2013, s. - [cit. 2013-05-4]. http://docs.basho.com/riak/latest/cookbooks/BasicClusterSetup/#AddaSecond-NodetoYourCluster

A Seznam příloh

[1] - grafy měření propustnosti databáze Cassandra - přiloženy na CD
[2] - výsledky měření propustnosti databází Cassandra a Riak - přiloženy na CD
[3] - programátorská dokumentace - přiložena na CD
[4] - uživatelská dokumentace - přiložena na CD